Milestone 5

Jesus Garnica

Lorenzo Moises

Alex Ha

Raul Serrano

Github and Milestone Links:

Github Repo

Milestone 1

Milestone 2

Milestone 3

Project Intro:

Divvy is an Android application where users can meet and connect to split the bill on any kind of service. Users can join a listing created by a user, allowing them to join chat room and negotiate splitting the bill for a service or item.

Proposed Scope:

1. Sign Up / Sign In / Sign Out
2. Search for services
3. Create a new listing
4. Add / Remove members to a listing
5. Close a listing
6. View other user listings
7. Conversate with other users
8. Send Image messages to users within a chat room specified to a listing
9. View user profiles
10. View user reviews on any user
11. Leave a review for a user

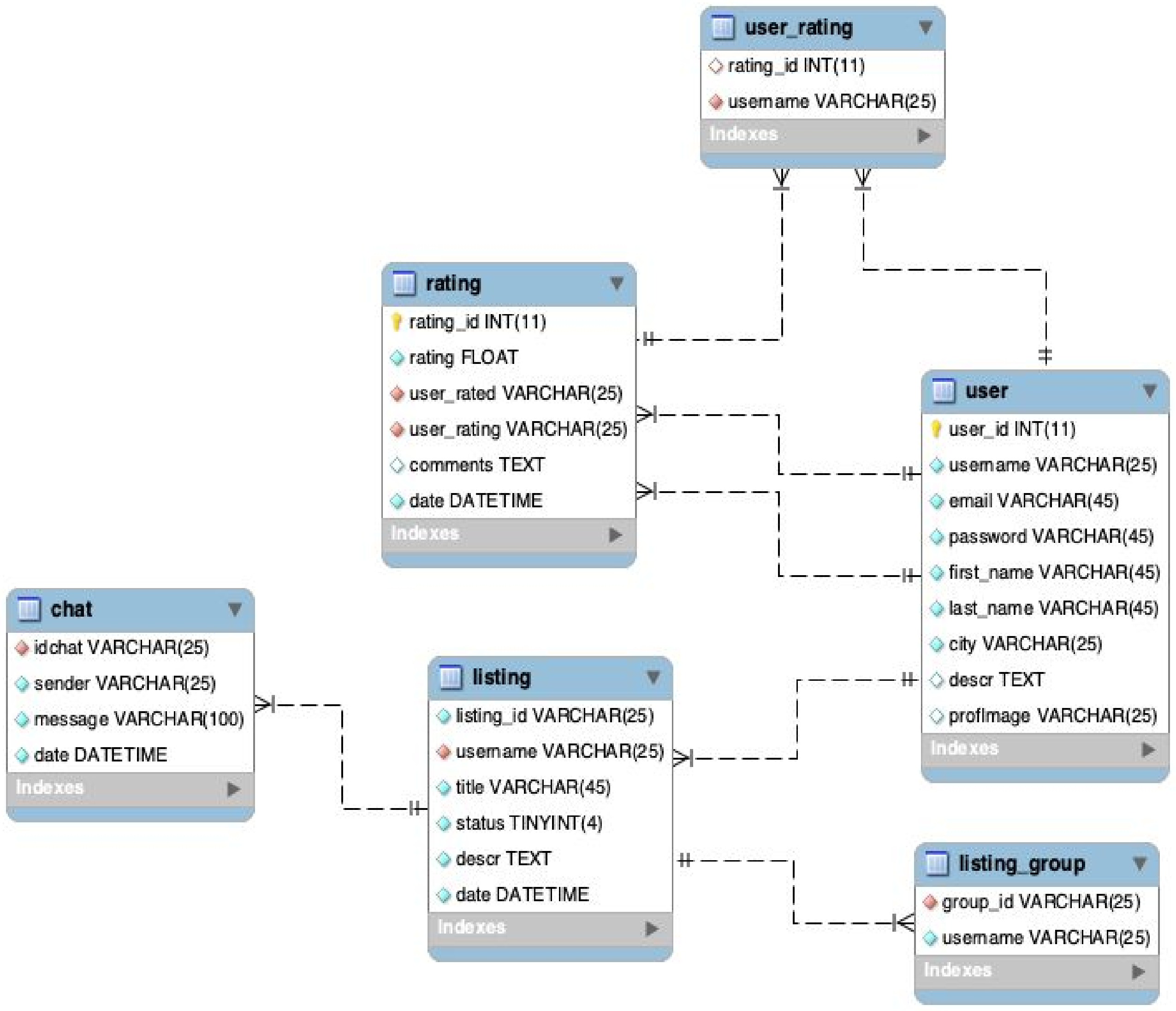
Current Scope:

1. Sign Up / Sign In / Sign Out
2. Search for services

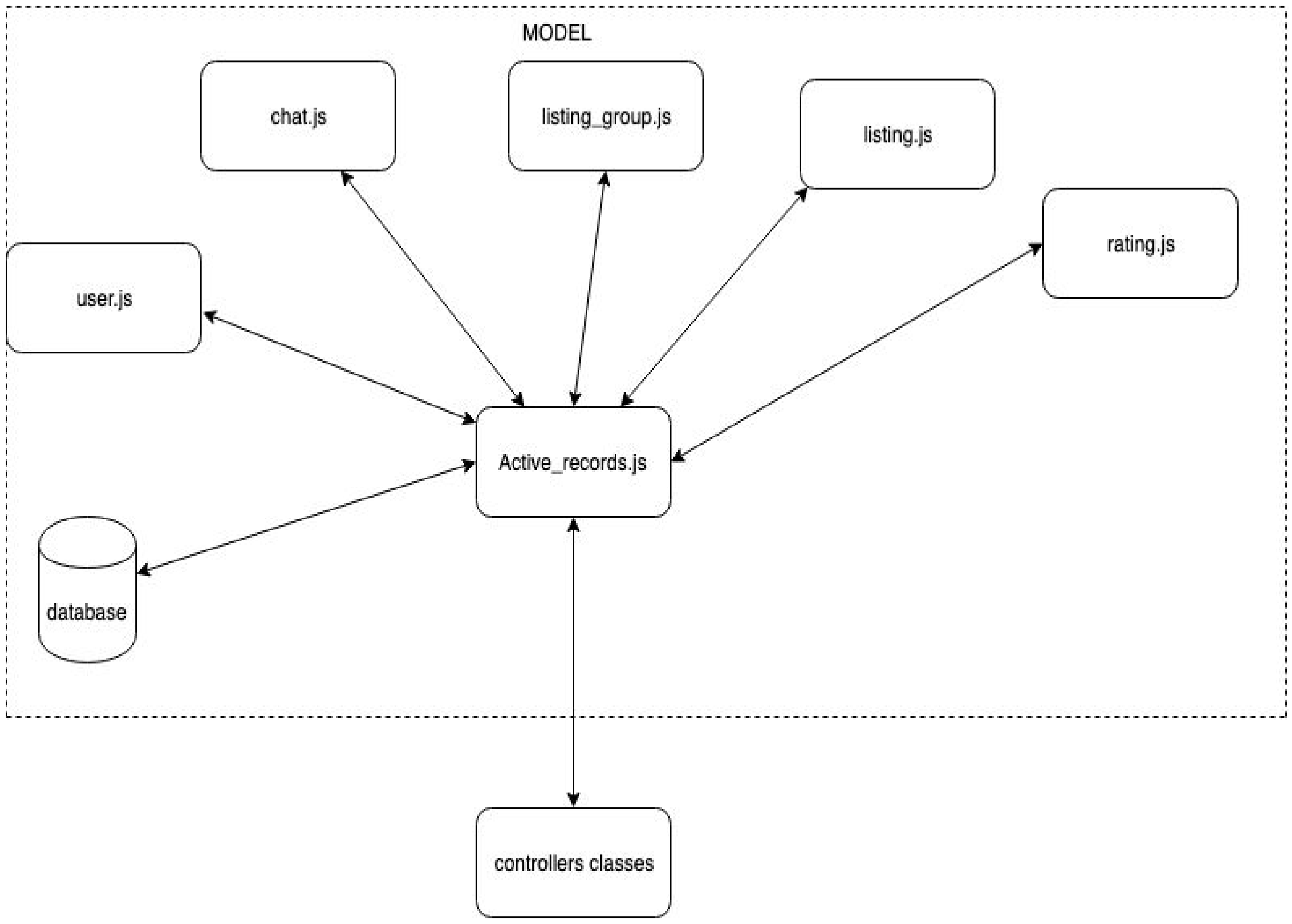
3. Create a new listing
4. Conversate with other users within a chat room specified to a listing
5. Send Image Messages to users within a chat room specified to a listing
6. View user profiles
7. View user reviews on any user
8. Leave a review for a user

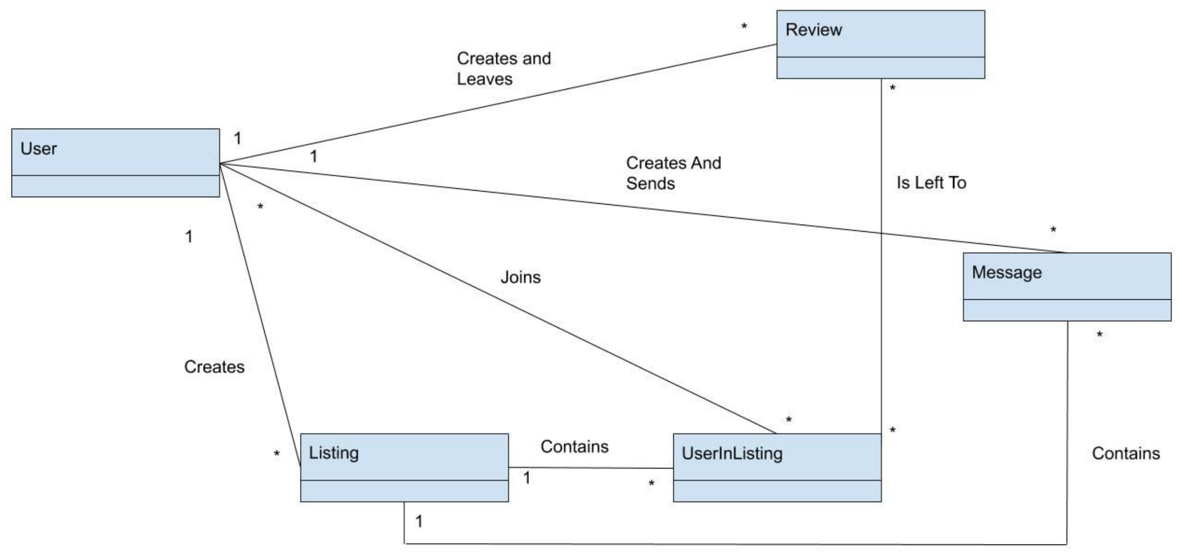
Scope of Work Described:

Conceptual Model: DATABASE



Conceptual Model: Backend





Use Cases:

Summary	The actor will download and open the app and register
Dependency	
Actor	Joe
Precondition	
Description	Joe downloads and opens the divvy app. He sees a login screen with an option to sign up. Joe clicks “signup” and is taken to another activity. In the signup view, joe inputs a username, first name, last name, email address, and password. Joe is taken to the login screen.
Alternative	Joe filled out the form incorrectly and is prompted to revise the form.
Postcondition	

Summary	The actor will download and open the app and register
Dependency	
Actor	Joe
Precondition	
Description	Joe downloads and opens the divvy app. He sees a login screen with an option to sign up. Joe clicks “signup” and is taken to another activity. In the signup view, joe inputs a username, first name, last name, email address, and password. Joe is taken to the login screen.
Alternative	Joe filled out the form incorrectly and is prompted to revise the form.
Postcondition	

Summary	The actor will download and open the app and register
Dependency	
Actor	Joe
Precondition	
Description	Joe downloads and opens the divvy app. He sees a login screen with an option to sign up. Joe clicks “signup” and is taken to another activity. In the signup view, joe inputs a username, first name, last name, email address, and password. Joe is taken to the login screen.
Alternative	Joe filled out the form incorrectly and is prompted to revise the form.
Postcondition	

Summary	The actor will download and open the app and register
Dependency	
Actor	Joe
Precondition	
Description	Joe downloads and opens the divvy app. He sees a login screen with an option to sign up. Joe clicks “signup” and is taken to another activity. In the signup view, joe inputs a username, first name, last name, email address, and password. Joe is taken to the login screen.
Alternative	Joe filled out the form incorrectly and is prompted to revise the form.
Postcondition	

Summary	The actor will download and open the app and register
Dependency	
Actor	Joe
Precondition	
Description	Joe downloads and opens the divvy app. He sees a login screen with an option to sign up. Joe clicks “signup” and is taken to another activity. In the signup view, joe inputs a username, first name, last name, email address, and password. Joe is taken to the login screen.
Alternative	Joe filled out the form incorrectly and is prompted to revise the form.
Postcondition	

System Design:

Model View Controller:

- Data Models:
    - User, Listing, Review, Message, Messenger
- Controllers:
    - CreateListingController, CreateReviewController, DetailedListingController, MessagingController, MyListingsController, NavBarController, SearchController, UserLoginController, UserProfileViewController, UserSignUpController
- Views (XML layout files):
    - activity_conversation, activity_create_listing, activity_create_view, activity_detailed_listings, activity_mylistings, activity search_controller, activity_user_login, activity_user_profile, fragment_listing, listings_view, message, review_view_holder, user_sign_up

Factory:

- RecyclerView
    - Contains a list of items (usually data models)
    - “Recycles” views when needed and not in use
    - Listing, reviews, and messages are displayed using a RecyclerView
- ViewHolder
    - One item in the RecyclerView (knows it’s order)
    - Multiple ViewHolders are used in Recycler View
    - Instantiate a subclass of this based on which model it is displaying
    - Each subclass displays a different view

State:

- Login Authenticator
    - Keeps track of user’s login state. It is also a singleton as mentioned above.

    - Allows us to persist the user login from the moment they login, to the moment they log out.
    - Stores data about the login into Android Shared preferences.
- Android Sharedpreferences
    - Stored in your app's data.
    - Data is deleted when you uninstall the app
    - Persistence of data between app sessions
    - Stores user login info to keep the user logged in when they leave the app.

Object-Oriented Discussion / Patterns: Backend

The backend has being designed with NodeJS, and Express following the Active record pattern and the Model View Controller pattern as much as possible. According to wikipedia "The active record pattern is an approach to accessing data in a database. A database table or view is wrapped into a class." in our project we have a class that connects to the database to insert and retrieve data and it is used by all the other classes in the model. Also we have created a class for each table that will be called upon request and they all inherits from the main Active-record class. This permits to have separated objects for each table and each request on the controller side. In the controller side we have post and get methods for each request that will call the model and wait for a response with the data retrieved from the database. This pattern is good for reusability because for each table we have a class that connects to one general class and does not reuse the same code over and over again which makes it easy to add new classes and new request without changing or affecting other classes and methods.

Object-Oriented Discussion / Patterns:

We decided early on that we were gonna turn all the entities into the objects that we could use in our code. Listing, message, user, and review are all the objects were gonna revolve the rest of the code on. We also made our design mainly focused around MVC with a few other patterns to help out certain elements.

Although Android is not the most perfect candidate for MVC, we found multiple ways to get this working. We created what we could into their objects that are only responsible for saving data as an object. We also made multiple controllers that were in charge of interacting with the view and linking up those models. We tried to make sure that the view had as little to do with the other component as possible. They often do not share code in between switching activities but rather they save it in somewhere else like sharedpreferences for user data for example. We created the controller to be in charge of getting data from the database and for sending that to be saved in the appropriate object.

We also created a few independent views that are created and reused in some views using the recycling adapter. The view got their data almost only from the objects created as to not break MVC. These views get used by the controller one at a time for the most part.

We also have a few factories that are used in very specific aspects of our code. We used the factory pattern for the messaging aspects of our application. These messages are used for a view used in the recyclerview and are all turned into their own objects. Recyclerview essentially takes a view and uses them as the basis to display in a list.

The observer pattern also played a key role since it is pretty crucial to MVC. When there is a state change or event, a listener will react to the changes. We put that to use in regard to listening for new messages, user action, and a few other areas.

Implementation Process:

We split the work based on what would be the most important to base the rest of the code on. We created the models that would be the foundation of our code. We placed these in the package labeled “models” with very clear names.

Conclusion

This project was very fun. The entire group was very responsive and were very engaged in development. Working in android studio was a pain but we followed our design patterns closely to produce a working product with OO design patterns and the MVC architectural pattern. Our team was smaller than the other ones but it was not too bad. Our app turned out pretty close to our proposed scope (minus/plus a few features).